LAB 6

Objective: Create a deployment, scale it, trigger a rolling update and rollback. Understand how replica sets allow you to manage revisions.

Pods are the lowest compute unit of Kubernetes. To ensure that pods are always running, you define a **deployment**. A deployment is a declarative manifest that describes what Pods should be running and how many.

To create your first deployment, use the **`kubectl run`** command. It is a convenience wrapper to define single container Pods. Let's create a deployment for *Ghost,* the microblogging platform:

```
$ kubectl run ghost --image=ghost
$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
ghost     1         1         1            1           6s
```

With your deployment running, check that a corresponding Pod has been started:

```
$ kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
ghost-943298627-kdhts    1/1       Running   0          9s
```

Now, scale your deployment. For example, let's scale it to five replicas:

```
$ kubectl scale deployments ghost --replicas=5
$ kubectl get pods
NAME                     READY     STATUS              RESTARTS   AGE
ghost-943298627-38761    0/1       ContainerCreating   0          2s
```



```
ghost-943298627-ghshb   0/1       ContainerCreating   0          2s
ghost-943298627-kdhts   1/1       Running             0          4m
ghost-943298627-slzdr   0/1       ContainerCreating   0          2s
ghost-943298627-xwljz   0/1       ContainerCreating   0          2s
```

To set the desired number of Pods, a replica sets resource has been automatically generated by the deployment. You can list this replica set with **`kubectl get rs`**. It continuously reconciles the desired state (e.g five replicas) with the actual cluster state. The replica set counts the number of Pods that matches the Pod selector defined in the deployment manifest and scales the Pods accordingly. Below, we show how you can parse the deployment manifest with **`jq`** to list the pod selection via labels.

```
$ kubectl get deployment ghost -o json | jq -r .spec.selector
{
  "matchLabels": {
    "run": "ghost"
  }
}
```

To see that the replica set does its job, delete one of the pods, and list Pods again. You will see that another one got started to still have the desired number of replicas running. You can also see this by removing the label from one of the running pods:

```
$ kubectl label pods ghost-943298627-38761 run-
$ kubectl get pods -Lrun
NAME                    READY     STATUS    RESTARTS   AGE       RUN
ghost-943298627-38761   1/1       Running   0          1h        <none>
ghost-943298627-ghshb   1/1       Running   0          1h        ghost
ghost-943298627-kdhts   1/1       Running   0          1h        ghost
ghost-943298627-nj122   1/1       Running   0          56m       ghost
ghost-943298627-slzdr   1/1       Running   0          1h        ghost
ghost-943298627-xwljz   1/1       Running   0          1h        ghost
```


Above you see that one Pod does not have the label **run=ghost**, because we removed it. The replica set automatically started a new Pod to have five running.

Explore a bit more the **kubectl labels** command and check how they can be set in resource metadata.

Rolling updates and rollbacks

One big advantage of deployments is that they can be used to do a rolling update of your Pods. For example, let's assume that you have a new image that you want to use. You can use the **kubectl set** command to change that image.

```
$ kubectl set image deployment/ghost ghost=ghost:0.9
```

List the replica sets again and see that a new set was created. This new set corresponds to the new image. The old and new replica sets will be scaled down and up respectively to keep the application running but provide the new image. At the end of the update, the new replica set has five running replicas and the old one has zero.

```
$ kubectl get rs
NAME              DESIRED   CURRENT   READY     AGE
ghost-3487275284  2         2         0         4s
ghost-943298627   4         4         4         1h
kubectl get rs
NAME              DESIRED   CURRENT   READY     AGE
ghost-3487275284  5         5         4         1m
ghost-943298627   0         0         0         1h
```

If you list your pods, you will see that the running pods now use the new replica set, except the pod that we re-labeled and was taken out of the replica set:

```
$ kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
ghost-3487275284-73lfn   1/1       Running   0          1m
ghost-3487275284-8pxvb   1/1       Running   0          2m
```



```
ghost-3487275284-pcbs5   1/1       Running   0          1m
ghost-3487275284-pnw4j   1/1       Running   0          2m
ghost-3487275284-t2b4q   1/1       Running   0          1m
ghost-943298627-38761    1/1       Running   0          1h
```

Now, check the history of your deployment:

```
$ kubectl rollout history deployment/ghost
deployments "ghost"
REVISION    CHANGE-CAUSE
1       <none>
2       <none>
```

You will see two revisions. The change-cause will be empty. You can record that cause by specifying the **`--record`** option when you create your deployment.

To rollback, simply do:

```
$ kubectl rollout undo deployment/ghost
```

If you keep an eye on your Pods, you will see that they will be terminated and new ones corresponding to the original replica set will be created. To learn more about deployments, do not forget to check the documentation.

